Chapter 6
Foundations for Program Development on CBE

Abstract In this chapter we present the foundations of parallel computing. We discuss Amdahl’s Law, Gustafson’s Law, the NC complexity class and Boolean circuits. We present pthreads, the POSIX thread API for expressing parallel programs. The multi-faceted parallelism present in the Cell architecture is introduced. We also briefly discuss OpenMP. This chapter also includes a list of things to watch out for when writing programs for the Cell.

6.1 Theory of Parallel Programming

We can classify parallelism as one of two categories: (a) control-flow parallelism and (b) data-parallel parallelism. Control flow parallelism is the most general model, but suffers from an inherent problem known as Amdahl’s law which limits the total speedup of the system as a function of its serial (or non-parallel) components. On the other hand, data-parallel is more restricted, but can yield very high levels of parallelism. There are now several models which implement control-flow parallelism on modern computers. The models are implemented using several techniques such as message passing, multiple instruction multiple data (MIMD), and locking. Shared memory multiprocessors use locking, while distributed computing based parallel systems use message passing.

Data-parallel (which is the favored model in this book as well) can be efficiently implemented using single-procedure multiple data (SPMD) or single-instruction multiple data. As we show later in this chapter, and in the sequel of this book, Synergistic Processor Units (SPUs) of the Cell implement data-parallel operations using both SIMD (at the instruction level) and SPMD (using DMA with multiple data packets being operated on by the SPE program)\(^1\)

\(^1\) In this book I use SPU and SPE interchangeably, but mostly when dealing with SIMD or instruction pipeline, floating-point, etc, I use SPU. When dealing with DMA, MailBoxes, synchronization, I refer to the unit as SPE.
In his book *Foundations of Parallel Programming* [81], Lewis describes the difference between control-flow analysis using Amdahl’s law and *Gustafson’s Law* for analyzing data-parallel parallelism. We describe the analysis below, but we should note, that both laws are derived in ideal settings where communication overheads (for example) are not analyzed in this simple model.

We first define *speedup* as the ratio of $T_1$ (which is the time taken by the program to complete on a single processor) to $T_n$ (which is the time taken by the program to complete on $n$ processors):

$$speedup = \frac{T_1}{T_n} \quad (6.1)$$

We define *efficiency* $E$ as a measure of the gains made by running it on $n > 1$ processors:

$$E = \frac{speedup}{n}$$

It used to be the case that at best one could hope for *linear* speedup, but with the advent of the *memory-wall*, and power-optimized CPU voltage scaling, we can actually achieve *super-linear* speedup (if the data is divided so that it becomes small enough to fit in the L2 cache, then program run-time can decrease by a factor of 100).

### 6.1.1 Amdahl’s Law

Amdahl’s law states that the time to run a parallel program on $n$ processors depends on the fraction of program $\beta$ which is serially executed and the $(1 - \beta)$ that is parallel.

$$T_n = \beta T_1 + \frac{T_1 (1 - \beta)}{n}$$

Substituting in Eqn. 6.1, we get

$$speedup = \frac{n}{\beta n + (1 - \beta)} \quad (6.2)$$

Consider a program which has $\beta = 0.6$ and a speedup factor of 10, then by Amdahl’s law the maximum speedup this program can achieve is:

$$s_{\text{max}} = \frac{1}{0.6 + \frac{1-0.6}{10}} = 1.5625$$

When we compare the component-wise speedup of 10, with the maximum achievable by the program as a whole, we can appreciate the struggle computer architects have had with Amdahl’s law, and the resulting architectures which have favored to speedup all parts, albeit, at the cost of ignoring performance headroom in other areas. SIMD optimization and multiprocessor have only recently been introduced in